**Remediation Component** - The Remediation Component is an expert system that facilitates integration of intelligent feedback into BusSim applications. It has the following features: Ability to compose high quality text feedback; Ability to compose multimedia feedback that includes video and/or audio; Ability to include reference material in feedback such as Authorware pages or Web Pages and Ability to actively manipulate the users deliverables to highlight or even fix users' errors. A proven remediation theory embedded in its feedback composition algorithm allows integration of digital assets into the Remediation of a training or IPS application. The Remediation model consists of three primary objects: Concepts; Coach Topics and Coach Items. ***Concepts*** are objects that represent real-world concepts that the user will be faced with in the interface. Concepts can be broken into sub-concepts, creating a hierarchical tree of concepts. This tree can be arbitrarily deep and wide to support rich concept modeling. Concepts can also own an arbitrary number of Coach Topics. ***Coach Topics*** are objects that represent a discussion topic that may be appropriate for a concept. Coach Topics can own an arbitrary number of Coach Items. ***Coach Items*** are items of feedback that may include text, audio, video, URL's, or updates to the Domain Model. Coach Items are owned by Coach Topics and are assembled by the Remediation Component algorithm.

**Workbenches-** The BusSim Toolset also includes a set of workbenches that are used by instructional designers to design and build BusSim applications. A ***workbench*** is a tool that facilitates visual editing or testing of the data that the BusSim Components use for determining an application's run-time behavior. The BusSim Toolset includes the following workbenches: **Knowledge Workbench** - The Knowledge Workbench is a tool for the creation of domain, analysis and feedback data that is used by the BusSim Components. It has the following features: Allows the designer to 'paint' knowledge in a drag-and-drop interface; Knowledge is represented visually for easy communication among designers; The interface is intelligent, allowing designers to only paint valid interactions; Designer's Task creations are stored in a central repository; The workbench supports check-in / check-out for exclusive editing of a task; Supports LAN-based or untethered editing; Automatically generates documentation of the designs; and it Generates the data files that drive the behavior of the components. **Simulated Student Test Workbench-** The Simulated Student Test Workbench is a tool for the creation of data that simulates student's actions for testing BusSim Component behaviors. It has the following features: The Test Bench generates a simulated application interface based on the Domain Model; The designer manipulates the objects in the Domain Model to simulate student activity; The designer can invoke the components to experience the interactions the student will experience in production; and The designer can fully test the interaction behavior prior to development of the application interface. **Regression Test Workbench** - The Regression Test Workbench is a tool for replaying and testing of student sessions to aid debugging. It has the following features: Each student submission can be individually replayed through the components; An arbitrary number of student submissions from the same session can be replayed in succession; Entire student sessions can be replayed in batch instantly; The interaction results of the student are juxtaposed with the results of the regression test for comparison.

**Development Cycle Activities**

The design phase of a BusSim application is streamlined by the use of the Knowledge Workbench. The Knowledge Workbench is a visual editor for configuring the objects of the component engines to control their runtime behavior. The components are based on proven algorithms that capture and implement best practices and provide a conceptual framework and methodology for instructional design. In conceptual design, the workbench allows the designer to paint a model of the hierarchy of Concepts that the student will need to master in the activity. This helps the designer organize the content in a logical way. The visual representation of the Concepts helps to communicate ideas to other designers for review. The consistent look and feel of the workbench also contributes to a streamlined Quality Assurance process. In addition, standard documentation can be automatically generated for the entire design. As the design phase progresses, the designer adds more detail to the design of the

Concept hierarchy by painting in Coach Topics that the student may need feedback on. The designer can associate multiple feedback topics with each Concept. The designer also characterizes each topic as being Praise, Polish, Focus, Redirect or one of several other types of feedback that are consistent with a proven remediation methodology. The designer can then fill each topic with text, video war stories, Web page links, Authorware links, or any other media object that can be delivered to the student as part of the feedback topic.

The toolset greatly reduces effort during functionality testing. The key driver of the effort reduction is that the components can automatically track the actions of the tester without the need to add code support in the application. Whenever the tester takes an action in the interface, it is reported to the domain model. From there it can be tracked in a database. Testers no longer need to write down their actions for use in debugging; they are automatically written to disk. There is also a feature for attaching comments to a tester's actions. When unexpected behavior is encountered, the tester can hit a control key sequence that pops up a dialog to record a description of the errant behavior. During the Execution Phase, the components are deployed to the student's platform. They provide simulated team member and feedback functionality with sub-second response time and error-free operation. If the client desires it, student tracking mechanisms can be deployed at runtime for evaluation and administration of students. This also enables the isolation of any defects that may have made it to production.

### Scenarios for Using the Business Simulation Toolset

A good way to gain a better appreciation for how the BusSim Toolset can vastly improve the BusSim development effort is to walk through scenarios of how the tools would be used throughout the development lifecycle of a particular task in a BusSim application. For this purpose, we'll assume that the goal of the student in a specific task is to journalize invoice transactions, and that this task is within the broader context of learning the fundamentals of financial accounting. A cursory description of the task from the student's perspective will help set the context for the scenarios. Following the description are five scenarios which describe various activities in the development of this task. The figure below shows a screen shot of the task interface. Figure 7 illustrates the use of a toolbar to navigate and access application level features in accordance with a preferred embodiment. A student uses a toolbar to navigate and also to access some of the application-level features of the application. The toolbar is the inverted L-shaped object across the top and left of the interface. The top section of the toolbar allows the user to navigate to tasks within the current activity. The left section of the toolbar allows the student to access other features of the application, including feedback. The student can have his deliverables analyzed and receive feedback by clicking on the Team button.

In this task, the student must journalize twenty-two invoices and other source documents to record the flow of budget dollars between internal accounts. (Note: "Journalizing", or "Journalization", is the process of recording journal entries in a general ledger from invoices or other source documents during an accounting period. The process entails creating debit and balancing credit entries for each document. At the completion of this process, the general ledger records are used to create a trial balance and subsequent financial reports.) In accordance with a preferred embodiment, an Intelligent Coaching Agent Tool (ICAT) was developed to standardize and simplify the creation and delivery of feedback in a highly complex and open-ended environment. Feedback from a coach or tutor is instrumental in guiding the learner through an application. Moreover, by diagnosing trouble areas and recommending specific actions based on predicted student understanding of the domain student comprehension of key concepts is increased. By writing rules and feedback that correspond to a proven feedback strategy, consistent feedback is delivered throughout the application, regardless of the interaction type or of the specific designer/developer creating the feedback. The ICAT is packaged with a user-friendly workbench, so that it may be reused to increase productivity on projects requiring a similar rule-based data engine and repository.

***Definition of ICAT In Accordance with a Preferred Embodiment***

The Intelligent Coaching Agent Tool (ICAT) is a suite of tools--a database and a Dynamic Link Library (DLL) run-time engine — used by designers to create and execute just-in-time feedback of Goal Based training. Designers write feedback and rules in the development tools. Once the feedback is set, the run-time engine monitors user actions, fires rules and composes feedback which describes the business deliverable. The remediation model used within ICAT dynamically composes the most appropriate feedback to deliver to a student based on student's previous responses. The ICAT model is based on a theory of feedback which has been proven effective by pilot results and informal interviews. The model is embodied in the object model and algorithms of the ICAT. Because the model is built into the tools, all feedback created with the tool will conform to the model. ICAT plays two roles in student training. First, the ICAT is a teaching system, helping students to fully comprehend and apply information. Second, ICAT is a gatekeeper, ensuring that each student has mastered the material before moving on to additional information. ICAT is a self contained module, separate from the application. Separating the ICAT from the application allows other projects to use the ICAT and allows designers to test feedback before the application is complete. The ICAT Module is built on six processes which allow a student to interact effectively with the interface to compose and deliver the appropriate feedback for a student's mistakes. ICAT development methodology is a seven step methodology for creating feedback. The methodology contains specific steps, general guidelines and lessons learned from the field. Using the methodology increases the effectiveness of the feedback to meet the educational requirements of the course. The processes each contain a knowledge model and some contain algorithms. Each process has specific knowledge architected into its design to enhance remediation and teaching. There is a suite of testing tools for the ICAT. These tools allow designers and developers test all of their feedback and rules. In addition, the utilities let designers capture real time activities of students as they go through the course. The tools and run-time engine in accordance with a preferred embodiment include expert knowledge of remediation. These objects include logic that analyzes a student's work to identify problem areas and deliver focused feedback. The designers need only instantiate the objects to put the tools to work. Embodying expert knowledge in the tools and engine ensures that each section of a course has the same effective feedback structure in place. A file structure in accordance with a preferred embodiment provides a standard system environment for all applications in accordance with a preferred embodiment. A development directory holds a plurality of sub-directories. The content in the documentation directory is part of a separate installation from the architecture. This is due to the size of the documentation directory. It does not require any support files, thus it may be placed on a LAN or on individual computers. When the architecture is installed in accordance with a preferred embodiment, the development directory has an _Arch, _Tools, _Utilities, Documentation, QED, and XDefault development directory. Each folder has its own directory structure that is inter-linked with the other directories. This structure must be maintained to assure consistency and compatibility between projects to clarify project differences, and architecture updates.

The _Arch directory stores many of the most common parts of the system architecture. These files generally do not change and can be reused in any area of the project. If there is common visual basic code for applications that will continuously be used in other applications, the files will be housed in a folder in this directory. The sub-directories in the _Arch directory are broken into certain objects of the main project. Object in this case refers to parts of a project that are commonly referred to within the project. For example, modules and classes are defined here, and the directory is analogous to a library of functions, APIs, etc... that do not change. For example the IcaObj directory stores code for the Intelligent Coaching Agent (ICA). The InBoxObj directory stores code for the InBox part of the project and so on. The file structure uses some primary object references as file directories. For example, the IcaObj directory is a component that contains primary objects for the ICA such as functional forms, modules and classes. The **BrowserObj** directory contains modules, classes and forms related to the browser functionality in the